Electronic Colloquium on Computational Complexity - Reports Series 1995 - available via:
ECCC
TR95-059
FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

The Monotone Theory for the PAC-Model

Nader H. Bshouty*
Department of Computer Science
The University of Calgary
Calgary, Alberta, Canada T2N 1N4
e-mail: bshouty@cpsc.ucalgary.ca
http://www.cpsc.ucalgary.ca/~bshouty/home.html

Abstract

We show that a DNF formula that has a CNF representation that contains at least one "$1/poly$-heavy" clause with respect to a distribution D is weakly learnable under this distribution. So DNF that are not weakly learnable under the distribution D do not have any "$1/poly$-heavy" clauses in any of their CNF representations. We then show that τ-CDNF, a DNF f that has a CNF representation that contains $poly(n)$ clauses that τ-approximates f according to a distribution D, is weakly $\tau+\epsilon$-PAC-learnable with membership queries under the distribution D.

We then show how to change our algorithm to a parallel algorithm that runs in poly-logarithmic time with a polynomial number of processors. In particular, decision trees are (strongly) PAC-learnable with membership queries under any distribution in parallel in poly-logarithmic time with a polynomial number of processors. Finally we show that no efficient parallel exact learning algorithm exists for decision trees.

1 Introduction

One of the outstanding open problems in computational learning theory is whether the class of polynomial size DNF is PAC-learnable in polynomial time with membership queries under any distribution D. Weak PAC-learning is learning that achieves an error that is different from $1/2$ (the guessing hypothesis) by $1/\text{poly}(n)$. Shapire [S90] showed that weak PAC-learning a class under any distribution implies (strong) PAC-learning of the class under any distribution. Therefore, finding a weak PAC-learning algorithm for DNF formulas under any distribution is as hard as finding a (strong) PAC-learning algorithm for DNF formulas under any distribution. Freund [F90,F92] showed that weak PAC-learning a class under any distribution D' that is poly away from D, i.e., satisfies $D/\text{poly}(n) \le D' \le D\text{poly}(n)$ implies PAC-learning under the distribution D. Jackson [J94] showed that DNF is weakly learnable under any distribution that is poly away from the uniform distribution and then using Freund's result gave a PAC-learning algorithm for

*This research was supported in part by NSERC of Canada.

DNF formulas under the uniform distribution that uses membership queries. Jackson showed that for every distribution that is poly away from the uniform distribution there is a boolean function g that can be found in polynomial time that agrees well with f. Then g can be used for the weak PAC-learning. The technique used by Jackson was the Fourier transform approach for learning.

In [Bs93] we used a different approach, the monotone theory, to show that any DNF is exactly learnable from membership and equivalence queries in time polynomial in the DNF and CNF size of the the target function. This implies PAC-learnability of CDNF formulas (poly size DNF and CNF) and decision trees with membership queries under any distribution.

In this paper we investigate weakly learning DNF via the monotone theory. We show that if there is a clause C_i in some CNF representation of the target f that is $1/poly$-heavy under the distribution D, i.e., $\Pr_D[C_i = 0] \geq 1/poly(n)$, then there is a weak learning algorithm for f. This shows that DNF that are (computationally) hard to weakly PAC-learn must not have a $1/poly$-heavy clause in any of its CNF representations. We also show that the class of functions f that can be τ-approximated by a small size CNF is weakly $\tau + \epsilon$-PAC learnable in polynomial time.

We then show that our algorithm can be changed to an algorithm that runs in parallel in polylogarithmic time with a polynomial number of processors. The time is polylogarithmic in the DNF size, the number of variables, $1/\epsilon$ and $1/\delta$. The number of processors is polynomial in the DNF size, the number of variables, $1/\epsilon$ and $1/\delta$. In particular, the class of decision trees is strongly PAC-learnable with membership queires under any distribution in parallel in polylogarithmic time with a polynomial number of processors.

Our algorithm uses the monotone theory [Bs93]. The sequential version of the algorithm in [Bs93] cannot be parallelized because many of the queries asked in the algorithm rely on the answer of the previous one. In this paper we develop a new version of the algorithm and show that it can be easily changed to a parallel algorithm. Our parallel algorithm also works for CDNF formulas and classes with known monotone basis (see [Bs93] for other classes). We also show that there exists no efficient exact learning algorithm for decision trees.

Our paper is organized as follows. In section 2 we define the learning models used in this paper. In section 3 we give the monotone theory for the exact and the PAC-learning models. In section 4 we give an algorithm for learning the monotone extention of a boolean function and in section 5 we give our algorithm and show how to change it to an efficient parallel algorithm. In section 6 we give the weak learning algorithm and then in section 7 we give the negative result for the parallel learning decision trees in the exact learning model.

2 The Model

The models considered in this paper are Exact learning from membership and equivalence queries [A88] and PAC-learning with membership queries under any distribution D [V84].

Let $\mathcal{C}$ and $\mathcal{H}$ be classes of boolean functions. In the Exact learning of $\mathcal{C}$ from $\mathcal{H}$ the learner wants to identify the *target* function $f \in \mathcal{C}$ using oracles for f. The learner knows $\mathcal{C}$, $\mathcal{H}$ and the number of variables of the target f. The oracles are membership and equivalence. To use the *membership oracle* the learner sends the oracle an assignment a and the oracle sends back the

value of f on a, i.e., $f(a)$. To use the *equivalence oracle* the learner sends the oracle a hypothesis $h \in \mathcal{H}$ and the equivalence oracle returns the answer "YES" if $f \equiv h$ and returns "(NO,c)", where $f(c) \neq h(c)$, otherwise. For sequential learning the goal of the learner is to run in time polynomial in the number of variables, n, the size of of the target function f ($size(f)$), and output some $h \in \mathcal{H}$ such that $h \equiv f$. For parallel exact learning the goal of the learner is to run in time polylogarithmic in n and $size(f)$ and output some $h \in \mathcal{H}$ such that $h \equiv f$.

For the PAC-learning, the learner receives examples of the target function $f \in \mathcal{C}$. The learner knows $\mathcal{C}$, $\mathcal{H}$, the number of variables of f and is given $\epsilon, \delta > 0$. The examples received by the learner are pairs $(x, f(x))$ where $x \in \{0,1\}^n$ is chosen according to the distribution D. The goal of the learner is to receive $poly(1/\delta, 1/\epsilon, n, size(f))$ examples and in $poly(1/\delta, 1/\epsilon, n, size(f))$ time, output a polynomial size circuit $h \in \mathcal{H}$ such that with probability at least $1 - \delta$

$$\Pr_D[f(x) \neq h(x)] \leq \epsilon.$$

In PAC-learning with membership queries the learner can also ask membership queries.

In the parallel PAC-learning the goal is to use $poly(1/\delta, 1/\epsilon, n, size(f))$ learners to learn such a hypothesis h in time $poly(\log(1/\delta), \log(1/\epsilon), \log n, \log size(f))$ time.

It is known from [A88] that Exact learning $\mathcal{C}$ from $\mathcal{H}$ implies that $\mathcal{C}$ is PAC-learnable with membership queries under any distribution.

3 The Monotone Theory

Here we will present the Monotone Theory developed in [Bs93] and prove other results that we will use for the correctness of our parallel algorithm.

3.1 The Monotone Theory for Exact Learning

In this section we give the Monotone Theory that is used in [Bs93] for exact learning decision trees. All the proofs of the lemmas can be found in [Bs93].

For a vector x representing an assignment to $\{X_1, \ldots, X_n\}$, $x[i]$ or x_i is the i-th entry of x. For two vectors we write $y \leq x$ if '$y[i] = 1$ implies $x[i] = 1$' for all i. For an assignment $a \in \{0,1\}^n$ we define $x \leq_a y$ if and only if $x + a \leq y + a$. Here $x + a$ is the group addition in $GF(2)^n$ (bitwise XOR).

A boolean function f is called a-monotone if $f(x+a)$ is monotone. The *(minimal) a-monotone boolean function* $\mathcal{M}_a(f)$ of f is defined as follows:

$$\mathcal{M}_a(f)(x) = \begin{cases} 1 & (\exists y \leq_a x)\ f(y) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

We write $\mathcal{M}$ for $\mathcal{M}_0$. The following are properties of $\mathcal{M}_a$.

Lemma 1 . We have

1. $\mathcal{M}_a(f) = \mathcal{M}(f(x+a))(x+a)$.

2. $\mathcal{M}_a(f \vee g) = \mathcal{M}_a(f) \vee \mathcal{M}_a(g)$.

3. $\mathcal{M}_a(f \wedge g) \Rightarrow \mathcal{M}_a(f) \wedge \mathcal{M}_a(g)$.

4. f is a-monotone iff $\mathcal{M}_a(f) = f$.

5. $f \Rightarrow \mathcal{M}_a(f)$.

6. If $f(a) = 1$ then $\mathcal{M}_a(f) \equiv 1$.

7. For a DNF $f = \bigvee_{i=1}^{k}(X_{e_{i,1}}^{c_{i,1}} \wedge \cdots \wedge X_{e_{i,\delta_i}}^{c_{i,\delta_i}})$ (here $X^0 = X$ and $X^1 = \bar{X}$) we have

$$\mathcal{M}_a(f) = \bigvee_{i=1}^{k} \bigwedge_{j:c_{i,j}=a[e_{i,j}]} X_{e_{i,j}}^{c_{i,j}}.$$

If all $c_{i,j} \neq a[e_{i,j}]$ for some term, then $\mathcal{M}_a(f) = 1$.

Another interesting property of the a-monotone boolean function of f is

Lemma 2 . We have

$$f \equiv \bigwedge_{a \in \{0,1\}^n} \mathcal{M}_a(f).$$

Definition 1 . Let $\mathcal{C}$ be a class of boolean functions. We define the monotone dimension $d = Mdim(\mathcal{C})$ of $\mathcal{C}$ to be the minimal number of assignments $a_1, \ldots, a_d$ such that for any $f \in \mathcal{C}$ we have

$$f \equiv \bigwedge_{i=1}^{d} \mathcal{M}_{a_i}(f).$$

A set of assignments $\{a_1, \ldots, a_d\}$ that satisfies the above equivalence for all $f \in \mathcal{C}$ is called an *M-basis* of $\mathcal{C}$ (d need not be minimal).

The following lemma shows a simple way to find the M-basis and Mdim of a class.

Lemma 3 . Let $\mathcal{C}$ be a class of boolean functions. Then $Mdim(\mathcal{C})$ is the minimal number of assignments $a_1, \ldots, a_d$ such that for every $f \in \mathcal{C}$, there exist d monotone boolean functions $M_1, \ldots, M_d$ such that

$$f \equiv M_1(x + a_1) \wedge \cdots \wedge M_d(x + a_d).$$

A set $\{a_1, \ldots, a_d\}$ that satisfies the above equivalence for any $f \in \mathcal{C}$ (with possibly different M_i) is an M-basis for $\mathcal{C}$.

This lemma implies the following corollary

Corollary 4. Let $f = C_1 \vee \cdots \vee C_s$ be a CNF where C_i, $i = 1, \ldots, s$ are clauses. Let A be a set of assignments such that for every $1 \leq i \leq s$ there is $a^{(i)} \in A$ where $C_i(a^{(i)}) = 0$. Then A is an M-basis for $\{f\}$ and therefore

$$f = \bigwedge_{a \in A} \mathcal{M}_a(f).$$

For a function f, the DNF size (CNF size) of f, $size_{DNF}(f)$, ($size_{CNF}(f)$), is the minimal number of terms (clauses) over all possible DNF (CNF) formulas of f. For a decision tree T the

decision tree size of T, $size_{DT}(T)$, is the number of leaves in the tree. The decision tree size of a boolean function f is

$$size_{DT}(f) = \min_{T \equiv f} size_{DT}(T).$$

Lemma 5 . We have

1. $size_{DNF}(\mathcal{M}(f)) \leq size_{DNF}(f)$.
2. $size_{DT}(f) \geq size_{DNF}(f) + size_{CNF}(f)$.
3. $Mdim(\{f\}) \leq size_{CNF}(f)$.

3.2 The Monotone Theory for the PAC-model

In this subsection we will develop the monotone theory for the PAC-learning model.

Definition 2. Let D be a distribution over $\{0,1\}^n$. A set of assignments A that satisfies

$$\Pr_D\left[f \neq \bigwedge_{a \in A} \mathcal{M}_a(f)\right] \leq \epsilon$$

is called a (D, ϵ)-M-basis for f.

The following lemma shows that if g is a CNF of size s that τ-approximates f and $f \Rightarrow g$ then a poly$(s, 1/\epsilon, 1/\delta)$ number of examples chosen according to the distribution D is an $(D, \tau+\epsilon)$-M-basis for f with probability at least $1-\delta$.

Lemma 6. Let D be any distribution over $\{0,1\}^n$. Let f and g be two boolean functions such that $f \Rightarrow g$, $g = C_1 \wedge \cdots \wedge C_s$ is a CNF and

$$\Pr_D[f = g] \geq 1 - \tau.$$

Let $A = \{a_1, \ldots, a_m\}$ be a set of assignments where $m = \lceil \frac{s}{\epsilon}(\ln s + \ln \frac{1}{\delta}) \rceil$ and each a_i is choosen according to the distribution D. Then with probability at least $1-\delta$ we have that A is a $(D, \epsilon+\tau)$-M-basis for f.

Proof. Suppose

$$\Pr_D[C_1 = 1] \leq \cdots \leq \Pr_D[C_r = 1] \leq 1 - \frac{\epsilon}{s} < \Pr_D[C_{r+1} = 1] \leq \cdots \leq \Pr_D[C_s = 1].$$

Consider the events

$$E_1 = [(\forall j \leq r)(\exists a_i \in A)\ C_j(a_i) = 0]$$

and

$$E_2 = \left[\Pr_D\left[f \neq \bigwedge_{a \in A} \mathcal{M}_a(f)\right] \leq \epsilon + \tau\right].$$

We will show

1. $E_1 \Rightarrow E_2$.

2. $\Pr[E_1] \geq 1 - \delta$.

Given those two properties we get

$$\Pr[E_2] \geq \Pr[E_1] \geq 1 - \delta$$

and the result follows.

To prove (1), suppose for every $j \leq r$ we have $C_j(a_{i(j)}) = 0$. Then by lemma 1

$$\begin{aligned} f &\Rightarrow \bigwedge_{a \in A} \mathcal{M}_a(f) \\ &\Rightarrow \bigwedge_{j=1}^{r} \mathcal{M}_{a_{i(j)}}(f) \\ &\Rightarrow \bigwedge_{j=1}^{r} \mathcal{M}_{a_{i(j)}}(C_j) \\ &= \bigwedge_{j=1}^{r} C_j. \end{aligned}$$

Therefore

$$\begin{aligned} \Pr_D\left[f \neq \bigwedge_{a \in A} \mathcal{M}_a(f)\right] &\leq \Pr_D\left[f \neq \bigwedge_{i=1}^{r} C_i\right] \\ &\leq \Pr_D[f \neq g] + \Pr_D[g \neq \wedge_{i=1}^{r} C_i] \\ &\leq \tau + \Pr_D\left[\bigwedge_{i=r+1}^{s} C_i = 0\right] \\ &\leq \tau + \frac{\epsilon}{s} s = \tau + \epsilon. \end{aligned}$$

Now to prove (2) we have

$$\begin{aligned} \Pr[E_1] &= 1 - \Pr[(\exists j \leq r)(\forall a_i \in A)\ C_j(a_i) = 1] \\ &\geq 1 - s\left(\Pr[C_j(a_i) = 1]\right)^m \qquad j \leq r \\ &\geq 1 - s\left(1 - \frac{\epsilon}{s}\right)^m \\ &\geq 1 - \delta. \square \end{aligned}$$

We now give a definition for $\mathcal{M}$ that suits the PAC-learning model.

Definition 3. The set $\mathcal{M}_a^{D,\eta}(f)$ is the set of all boolean functions g such that

1. $\Pr_D[g \wedge f \neq f] \leq \eta$.
2. $g \Rightarrow \mathcal{M}_a(f)$.

Notice that in this definition a function $g \in \mathcal{M}_a^{D,\eta}(f)$ is not necessary an η-approximation of $\mathcal{M}_a(f)$. Now we will show that if $g_i \in \mathcal{M}_{a_i}^{D,\eta}(f)$ and $h = \wedge_i \mathcal{M}_{a_i}(f)$ is an approximation for f then $\wedge_i g_i$ is an approximation for f.

Lemma 7. Let $g_i \in \mathcal{M}_{a_i}^{D,\eta}(f)$ for $i = 1, \ldots, r$. If $A = \{a_i | i = 1, \ldots, r\}$ is a (D, ξ)-M-basis of f then

$$\Pr_D \left[f \neq \bigwedge_{i=1}^{r} g_i \right] \leq \xi + r\eta.$$

Proof. Let $h = \bigwedge_{i=1}^{r} \mathcal{M}_{a_i}(f)$ and $g = \bigwedge_{i=1}^{r} g_i$. Since

$$\begin{aligned} g \wedge h &= \bigwedge_{i=1}^{r} (g_i \wedge \mathcal{M}_{a_i}(f)) \\ &= \bigwedge_{i=1}^{r} g_i \\ &= g \end{aligned}$$

we have

$$\begin{aligned} \Pr_D[g \neq f] &= \Pr_D[g \neq g \wedge f] + \Pr_D[g \wedge f \neq f] \\ &\leq \Pr_D[g \wedge h \neq g \wedge f] + \sum_{i=1}^{r} \Pr_D[(g_i \wedge f) \neq f] \\ &\leq \Pr_D[h \neq f] + r\eta \\ &\leq \xi + r\eta. \square \end{aligned}$$

Now the following theorem combines lemma 6 and 7.

Theorem 8. Let $f \Rightarrow g$ be boolean functions where g has a CNF of size s. Let D be a distribution over $\{0,1\}^n$ and A be a set of

$$m = \left\lceil \frac{2s}{\epsilon} \left(\ln s + \ln \frac{1}{\delta} \right) \right\rceil$$

assignments choosen according to the distribution D. Let B be the set of assignments $a \in A$ that satisfy $f(a) = 0$. For every $b \in B$ let $g_b \in \mathcal{M}_b^{D,\eta}(f)$ where

$$\eta = \frac{\epsilon}{2m}.$$

Then with probability at least $1 - \delta$ we have

$$\Pr_D \left[f \neq \bigwedge_{b \in B} g_b \right] \leq \epsilon + \Pr_D[f \neq g].$$

Proof. First notice that if $a \in A$ and $f(a) = 1$ then $\mathcal{M}_a(f) = 1 \in \mathcal{M}_a^{D,\eta}(f)$ for any D and η. Therefore we may eliminate all positive examples in A.

Now by lemma 6, with probability at least $1-\delta$, B is an $(D, \epsilon/2 + \Pr_D[f \neq g])$-M-basis and by lemma 7 we get

$$\begin{aligned} \Pr_D\left[f \neq \bigwedge_{b\in B} g_b\right] &\leq \frac{\epsilon}{2} + \Pr_D[f \neq g] + m\frac{\epsilon}{2m} \\ &\leq \epsilon + \Pr_D[f \neq g].\Box \end{aligned}$$

Now the problem of learning f is reduced to learning some $g \in \mathcal{M}_a^{D,\eta}$ from examples of f. We will show this in the next section.

4 Learning $h \in \mathcal{M}_a^{D,\eta}(f)$

In this section we will show how to learn a $h \in \mathcal{M}_a^{D,\eta}(f)$. Recall that $h \in \mathcal{M}_a^{D,\eta}(f)$ if the following two conditions hold.

1. $\Pr_D[h \wedge f \neq f] \leq \eta$.
2. $h \Rightarrow \mathcal{M}_a(f)$.

Notice that if only the first condition is required we could just give $h = 1$. The next Theorem shows that after taking enough examples according to the distribution D any small size hypothesis that is consistent on **only** the positive examples will satisfy condition 1. Later in this section we will show how to make sure that our hypothesis also satisfies condition 2.

Lemma 9. Let f be a boolean function. Suppose there is an algorithm $\mathcal{A}$ such that for every assignment $a_1, \ldots, a_t \in \{0,1\}^n$ that satisfies $f(a_1) = \cdots = f(a_t) = 1$ it finds a function $h = \mathcal{A}(a_1, \ldots, a_t)$ in some set of functions $\mathcal{H}$ where $h(a_1) = \cdots = h(a_t) = 1$. Let $b_1, \ldots, b_s \in \{0,1\}^n$ be randomly chosen according to the distribution D where

$$s = \left\lceil \frac{1}{\eta}\left(\ln|\mathcal{H}| + \ln\frac{1}{\delta}\right)\right\rceil.$$

Then for $B = \{b_i | f(b_i) = 1\}$ and $h = \mathcal{A}(B)$ with probability at least $1-\delta$ we have

$$\Pr_D[h \wedge f \neq f] < \eta.$$

Proof. Let B_η be the set of all $g \in \mathcal{H}$ such that

$$\Pr_D[g \wedge f \neq f] > \eta.$$

Then

$$\begin{aligned} \Pr[\Pr_D[h \wedge f \neq f] > \eta] &= \Pr[(\exists h \in B_\eta)\ h \wedge f \neq f \text{ on } b_i] \\ &\leq |B_\eta| \Pr[h \wedge f \neq f \text{ on } b_i] \quad h \in B_\eta \\ &\leq |\mathcal{H}| \prod_i \Pr[(h \wedge f)(b_i) \neq f(b_i)] \\ &\leq |\mathcal{H}|(1-\eta)^s \\ &\leq \delta.\Box \end{aligned}$$

A better result can be obtained using the $VCdim$ but for our algorithm the $VCdim$ will give the same bound.

We now show how to learn a hypothesis h that satisfies both conditions:

1. $\Pr_D[h \wedge f \neq f] \leq \eta$.
2. $h \Rightarrow \mathcal{M}_a(f)$.

Let $a_1, \ldots, a_s$ be examples that are given according to the distribution D. First note that any algorithm that learns an $h \in \mathcal{M}^{D,\eta}(f)$ can be changed to an algorithm that learns an $h' \in \mathcal{M}_a^{D,\eta}(f)$. Just run the first algorithm for the examples $a_i + a$ to get a hypothesis h and then return the hypothesis $h' = h(x + a)$. Therefore, it is enough to give an algorithm that learns an $h \in \mathcal{M}^{D,\eta}(f)$.

First we will ignore all the negative examples. Let $B = \{a_i | f(a_i) = 1\}$. Suppose f is a DNF of size t. As long as $|B| > 2t$ we build the following graph. The graph is $G_f(B)$. The nodes of the graph are the assignments in B and the edges satisfy $(a, b) \in E$ if and only if $f(a \wedge b) = 1$. Notice that this is the stage where we use the membership oracle to find $f(a \wedge b)$ and build the graph. We find a maximum matching $M \subset E$ in the graph $G_f(B)$ and take each edge $(a, b) \in M$ and collapse the two nodes a and b to a new node $a \wedge b$. The set of nodes of the new graph is denoted by $B[M]$. We recursively do the above until the number of nodes in the graph is at most $2t$. Let $B^\star$ be the final set of nodes. Now the hypothesis will be

$$h_{B^\star} = \bigvee_{b \in B^\star} T^b$$

where

$$T^b = \bigwedge_{b_i = 1} X_i.$$

We now prove a sequence of results

The following lemma shows that after $O(\log s)$ matchings the algorithm stops.

Lemma 10. Let B be a set of positive examples of a DNF f of size t. Let M be a maximum matching in $G_f(B)$. If $|B| \geq 2t$ then

$$|B[M]| \leq \frac{3|B|}{4}.$$

Proof. Let $f = T_1 \vee \cdots \vee T_t$. Let $B = B_1 \cup \cdots \cup B_t$ be a partition of B such that each $a \in B_i$ satisfies the term T_i. Notice that for every $a, b \in B_i$, (a, b) is an edge in the graph $G_f(B)$. This is because if $T_i(a) = T_i(b) = 1$ then $T_i(a \wedge b) = 1$ and therefore $f(a \wedge b) = 1$. Therefore $G_f(B_i)$ is a clique for every i. Since a clique of size k has a matching of size $\lfloor k/2 \rfloor$ the graph $G_f(B)$ will have a maximum matching of size at least

$$\lfloor |B_1|/2 \rfloor + \cdots + \lfloor |B_t|/2 \rfloor \geq \frac{|B|}{2} - \frac{t}{2}.$$

Therefore the number of nodes in $B[M]$ is at most

$$\frac{|B|}{2} + \frac{t}{2} \leq \frac{3|B|}{4}.\Box$$

Lemma 11. The hypothesis $h_{B^\star}$ satisfies condition 2. That is

$$h_{B^\star} \Rightarrow \mathcal{M}(f).$$

Proof. Notice that the algorithm starts from set of assignments B that satisfy f. Since the algorithm collapses two nodes a and b only when $f(a \wedge b) = 1$ we have that for every element in $B^\star$, $f(b) = 1$. Therefore $T^b \Rightarrow \mathcal{M}(f)$ for all $b \in B^\star$ and therefore

$$h_{B^\star} = \bigwedge_{b \in B^\star} T^b \Rightarrow \mathcal{M}(f).\Box$$

Lemma 12. If the number of examples in the algorithm is

$$s = \left\lceil \frac{1}{\eta}\left((2\ln 2)nt + \ln\frac{1}{\delta}\right)\right\rceil$$

then with probability at least $1-\delta$ we have

$$\Pr_D[h_{B^\star} \wedge f \neq f] \leq \eta.$$

Proof. We apply lemma 9. Since the output hypothesis of the algorithm is a monotone DNF with at most $2t$ terms and since the number of monotone DNF with at most $2t$ terms is at most 2^{2nt} we get the result.$\Box$

Lemma 11 and 12 shows that

Theorem 13. The above algorithm with

$$s = \left\lceil \frac{1}{\eta}\left((2\ln 2)nt + \ln\frac{1}{\delta}\right)\right\rceil$$

examples returns a hypothesis h that with probability at least $1-\delta$ is in $\mathcal{M}_a^{D,\eta}(f)$.

5 The Sequential and Parallel Algorithm

We are now ready to introduce the sequential and parallel algorithm. We will combine theorems 8 and 13 to obtain the algorithm in figure 1.

Theorem 14. Let τ-CDNF be the class of DNF f that have CNF, $C_1 \wedge C_2 \wedge \cdots$ where $g = C_1 \wedge \cdots \wedge C_s$ is a τ-approximation of f. For any f that is a τ-CDNF, algorithm **τ-CDNF-Algorithm** runs using

$$O\left(\frac{s}{\epsilon^2}\left(size_{DNF}(f)n + \ln\frac{1}{\epsilon} + \ln^2 s + \ln^2\frac{1}{\delta}\right)\right)$$

τ-CDNF-Algorithm

1) Take $m = \lceil \frac{2s}{\epsilon}\left(\ln s + \ln \frac{2}{\delta}\right) \rceil$ examples $a_1, \ldots, a_m$.
2) $A \leftarrow \{a_i | f(a_i) = 0\}$.
3) If $B = \emptyset$ then return(1).
4) Take $r = \lceil \frac{2m}{\epsilon}\left((2\ln 2)nt + \ln \frac{m}{\delta}\right) \rceil$ examples $b_1, \ldots, b_r$.
5) $B \leftarrow \{b_i | f(b_i) = 1\}$.
6) If $B = \emptyset$ then return(0).
7) For each $b \in A$ do the following.
   7.1) $B_b \leftarrow \{b + b_i | b \in B\}$.
   7.2) While $|B_b| \geq 2t$ do the following.
      7.2.1) Define the graph $G_f(B) = (B, E)$.
      7.2.2) $E \leftarrow \{(b^{(1)}, b^{(2)}) | b^{(1)}, b^{(2)} \in B_b,\ f(b^{(1)} \wedge b^{(2)}) = 1\}$.
      7.2.3) Find a maximum matching M in $G_f(B)$.
      7.2.4) For each $(e_1, e_2) \in M$ do $B_b \leftarrow (B_b \backslash \{e_1, e_2\}) \cup \{e_1 \wedge e_2\}$.
   7.3) $h'_{B_b} \leftarrow \bigwedge_{b' \in B_b} \left(\bigvee_{b'[i]=1} X_i \right)$.
   7.4) $h_{B_b} \leftarrow h'_{B_b}(x + b)$.
8) Return$\left(h = \bigwedge_{b \in B} h_{B_b}\right)$.

Figure 1: *Algorithm for learning τ-CDNF.*

examples and poly$(n, size_{DNF}(f), s, 1/\epsilon, 1/\delta)$ time and outputs a depth two hypothesis h that with probability at least $1-\delta$ satisfies

$$\Pr_D[f = h] \geq 1 - \tau - \epsilon.$$

Proof. This theorem follows immediately from theorems 8 and 13.□

Next we will show that this algorithm can be changed to an efficient parallel learning algorithm. We will show that each step in the algorithm can be executed efficiently in parallel. In the algorithm we use $m = (2s/m)(\ln s + \ln(2/\delta))$ processors to execute step 1. Each processor takes one example and stays active if and only if its example is negative. If no process stays active then the algorithm returns 1. Now the algorithm proceeds in step 4 by taking r examples (see the algorithm) and constructs the set B. This can be done in NC^1. Now each active processor that corresponds to $b \in A$ will run r processors and define B_b. The processors corresponding to b will build the graph $G_f(B)$ and find a maximal matching. In [MVV87] it is shown that maximal matching in graphs can be done in RNC^2. We also showed in section 4 that the number of matching we need is $\log s$ and therefore step 7 can be done efficiently in parallel. In steps 7.3, 7.4 and 8 each processor can build the hypothesis in parallel and output h.

In particular we have the following.

Theorem 16. The class of decision trees is PAC-learnable with membership queries under any distribution in polylogarithmic time with polynomial number of processors.

See [Bs93] for more classes.

6 Weakly Learning DNF

In this section we prove the following.

Theorem 15. Let $\mathcal{C}_k$ be the class of DNF f whose CNF representation $f = C_1 \wedge C_2 \wedge \cdots$ contains a clause C_i with

$$\Pr_D[C_i = 0] \geq \frac{1}{n^k}.$$

Then $\mathcal{C}_k$ is $\frac{1}{2} - \frac{1}{4n^k}$-weakly PAC-learnable with membership queries under the distribution D in polynomial time.

Proof. Consider the algorithm in Figure 2.

If the execution of the algorithm passes steps 1) and 2) then we have

$$\frac{1}{2} - \frac{1}{2n^k} \leq \Pr_D[f = 0] \leq \frac{1}{2} + \frac{1}{2n^k}.$$

If we randomly choose a then with probability at least $1 - \delta = 1/n^k$ we get an assignment that satisfies $C_i(a) = 0$. For such an assignment we have

$$f \Rightarrow \mathcal{M}_a(f) \Rightarrow \mathcal{M}_a(C_i) = C_i.$$

Week-Algorithm

1) If $\Pr_D[f = 0] \geq \frac{1}{2} + \frac{1}{2n^k}$ then return(0).
2) If $\Pr_D[f = 1] \geq \frac{1}{2} + \frac{1}{2n^k}$ then return(1).
3) Randomly choose a and learn $h \in \mathcal{M}_a^{D, \frac{1}{4n^k}}$.
4) Output(h).

Figure 2: *Week Learning DNF.*

Therefore,

$$\begin{aligned}
\Pr_D[f \neq h] &\leq \Pr_D[f \neq h \wedge f] + \Pr_D[h \wedge f \neq h] \\
&\leq \frac{1}{4n^k} + \Pr_D[h \wedge f \neq h \wedge \mathcal{M}_a(f)] \\
&\leq \frac{1}{4n^k} + \Pr_D[f \neq \mathcal{M}_a(f)] \\
&\leq \frac{1}{4n^k} + \Pr_D[f \neq C_i] \\
&\leq \frac{1}{4n^k} + \Pr_D[f = 0] \Pr_D[C_i = 1 | f = 0] \\
&\leq \frac{1}{4n^k} + \left(\frac{1}{2} + \frac{1}{2n^k}\right)\left(1 - \frac{1}{n^k}\right) \\
&\leq \frac{1}{2} - \frac{1}{4n^k}. \Box
\end{aligned}$$

7 Lower Bound for Exact Learning

In this section we show that decision trees cannot be exactly learned in parallel in polylogarithmic time. We will use the following result that is proven in [BC92].

Let $e(m)$ be the number of equivalence queries needed to learn the class of polynomial size decision trees with unlimited computational time when m membership queries are allowed to be asked in the algorithm. Let $ME(p)$ be the number of parallel steps needed to learn polynomial size decision trees with unlimited computational time and p processors.

Lemma 17 [BC92]. We have

$$ME\left(\frac{m}{e(m)}\right) \geq e(m).$$

In this section we will show the following.

Lemma 18. We have

$$e(m) \geq \Omega\left(\frac{n}{\log m}\right).$$

Since $e(poly(n)) \leq poly(n)$ (use, for example, the halving algorithm) we get the next theorem.

Theorem 19. We have

$$ME(poly(n)) \geq \Omega\left(\frac{n}{\log n}\right).$$

That is, the number of parallel steps to exactly learn polynomial size decision trees with polynomial number of processors is at least $n/\log(n)$.

Proof of Lemma 18. Let $\mathcal{A}$ be an algorithm that learns decision trees with m membership queries and e equivalence queries. We run the algorithm $\mathcal{A}$ with the following adversary. The possible hypotheses that the adversary uses are the set of decision trees that are of the form

```
if (x_1, ..., x_k) = (ξ_1, ..., ξ_k) then
 if x_{k+1} = 1 then output 1
 elseif x_{k+1} = 0 then if (x_{k+2}, ..., x_{2k+1}) = (ξ_{k+2}, ..., ξ_{2k+1}) then
 if x_{2(k+1)} = 1 then output 1
  elseif ......
⋮          ⋮
      elseif x_{s(k+1)} = 0 then if (x_{s(k+1)+1}, ..., x_{s(k+1)+k}) = (ξ_{s(k+1)+1}, ..., ξ_{s(k+1)+k}) then
      if x_{(s+1)(k+1)} = 1 then output 1
        else output(0).
```

Notice that this function is 1 for an assignment a if and only if the prefix of the assignment a is of the form

$$\xi_1, \ldots, \xi_k, 0, \xi_{k+2}, \ldots, \xi_{2k+1}, 0, \ldots\ldots, \xi_{r(k+1)+1}, \ldots, \xi_{r(k+1)+k}, 1.$$

The size of the above decision tree is at most n. The adversary chooses $k = \lceil \log m \rceil + 1$ and $s + 1 = \lfloor n/(k+1) \rfloor$.

The adversary will define sets $S_0, \ldots, S_s = \{0,1\}^k$ where S_i is a set of possible values for $(\xi_{i(k+1)+1}, \ldots, \xi_{i(k+1)+k})$.

The adversary starts by answering 0 for every membership query $a = (a_1, \ldots, a_n)$ that the algorithm asks and it removes $(a_1, \ldots, a_k)$ from S_0. Notice that since the algorithm $\mathcal{A}$ asks at most m membership queries and since $|S_0| = 2^k > m$ the adversary will never run out of assignments in S_0. The algorithm then asks an equivalence query EQ(h). If $h \not\equiv 0$ then the adversary returns any assignment that satisfies $h(a) = 1$ and removes $(a_1, \ldots, a_k)$ from S_0. This equivalence query will not contribute much to the knowledge of the learning algorithm. The only equivalence query that will help the learning algorithm is the one that forces the adversary to determine $(\xi_1, \ldots, \xi_k)$. When the algorithm asks EQ(0) the adversary will return any assignment $(\xi_1, \ldots, \xi_k, 1, 0, 0, \ldots, 0)$ for some $(\xi_1, \ldots, \xi_k) \in S_0$.

So far the learning algorithm knows only $(\xi_1, \ldots, \xi_k)$ from one equivalence query and knows nothing about the other ξs. After $r+1$ equivalence queries the learning algorithm knows

$$(\xi_1, \ldots, \xi_k), (\xi_{k+2}, \ldots, \xi_{2k+1}), \ldots, \ldots, (\xi_{r(k+1)+1}, \ldots, \xi_{r(k+1)+k}).$$

The adversary will proceed as follows. For every membership query $a = (a_1, \ldots, a_n)$ that the learning algorithm asks, if the prefix of the assignment is not

$$\xi^{(1)} = \xi_1, \ldots, \xi_k, 0, \xi_{k+2}, \ldots, \xi_{2k+1}, 0, \ldots, \ldots, \xi_{r(k+1)+1}, \ldots, \xi_{r(k+1)+k}, 0$$

then the answer to the membership query can be uniquely determined by the hypothesis learnt already by the learning algorithm and the learning algorithm can gain no information about the target. If the prefix of the assignment of a agrees with the above then the adversary returns 0 and removes $(a_{(r+1)(k+1)+1}, \ldots, a_{(r+1)(k+1)+k})$ from S_{r+1}. As before m membership queries cannot find $(\xi_{(r+1)(k+1)+1}, \ldots, \xi_{(r+1)(k+1)+k})$. Therefore the learning algorithm must ask an equivalence query. Let h be the hypothesis of the equivalence query. If $h(a) = 1$ for some a with the prefix $\xi^{(1)}$ then the adversary returns a as a couterexample and removes $(a_{(r+1)(k+1)+1}, \ldots, a_{(r+1)(k+1)+k})$ from S_r. Otherwise, the adversary is forced to reveal $(\xi_{(r+1)(k+1)+1}, \ldots, \xi_{(r+1)(k+1)+k})$. It chooses some $(\xi_{(r+1)(k+1)+1}, \ldots, \xi_{(r+1)(k+1)+k}) \in S_{r+1}$ and returns the counterexample

$$(\xi_1, \ldots, \xi_k, 0, \xi_{k+2}, \ldots, \xi_{2k+1}, 0, \ldots, \ldots, \xi_{(r+1)(k+1)+1}, \ldots, \xi_{(r+1)(k+1)+k}, 1, 0, 0, \ldots, 0).$$

It is now clear that the number of equivalence query that the learning algorithm needs to ask is

$$s + 1 \geq \frac{n}{\lceil \log m \rceil + 3} = \Omega\left(\frac{n}{\log m}\right). \square$$

References

[A88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[Bs93] N. H. Bshouty. Exact learning via the monotone theory. In *Proceedings of the 34th Symposium on Foundations of Computer Science*. pages 302–311, November 1993.

[BC92] N. H. Bshouty and R. Cleve. On the exact learning of formulas in parallel, Proc. of the 33rd Symposium on the Foundations of Comp. Sci., IEEE Computer Society Press, Los Alamitos, CA, 1992, 513–522.

[F90] Y. Freund. Boosting a weak learning algorithm by majority, Proc. 3rd Annu. Workshop on Comput. Learning Theory, San Mateo, CA, 1990, 202–216.

[F92] Y. Freund. An improved boosting algorithm and its implications on learning complexity, Proc. 5th Annu. Workshop on Comput. Learning Theory, ACM Press, New York, NY, 1992, 391–398.

[J94] J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. In *Proceeding of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[MVV87] K. Mulmuley, U. V. Vazirani, V. V. Vazirani. Matching is as Easy as Matrix Inversion, ACM Symposium on Theory of Computing (STOC), 1987, 345–354.

[S90] R. E. Schapire, The Strength of Weak Learnability, *Machine Learning,* 197–227, 5, 2, 1990.

[Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.